Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Monitoraggio degli alveari e prevenzione della sciamatura delle api attraverso l'utilizzo di sistemi IoT

Tesi di Laurea in:
SISTEMI EMBEDDED E INTERNET OF THINGS

Relatore:
Prof. ALESSANDRO RICCI

Presentata da:
ROBERTO DRAGHETTI

Correlatore:
STEFANO RIGHINI

Anno Accademico 2022-2023

Indice

Capitolo 1

Introduzione

1.1 L'apicoltura oggi

Le api sono animali estremamente importanti per noi: sono responsabili di circa il 70% dell'impollinazione delle piante a livello globale, e sono indirettamente coinvolte in circa il 35% della produzione di cibo di ogni genere[2]. La sopravvivenza di questa specie è estremamente importante non solo per la produzione di miele, ma per l'intero **equilibrio dell'ecosistema**.

Negli ultimi anni, tuttavia, si sta assistendo ad un calo del numero degli esemplari di api in alcune zone del mondo, dovuto principalmente a pesticidi, parassiti e, soprattutto, al cambiamento climatico. In particolare, i fenomeni climatici estremi, come siccità, alluvioni, aumento delle temperature e non solo, stanno causando sempre più difficoltà nel campo dell'apicoltura: nel 2023, in Italia la produzione di miele ha visto un calo, in media, del 75%.

In più, l'apicoltura rimane oggi uno dei mestieri che richiedono maggiormente la sorveglianza e l'intervento umano, con la quasi impossibilità di automatizzazione del processo.

Il risultato più evidente di quello che sta succedendo all'apicoltura nel nostro paese è il crescente bisogno di **importazione** di miele dall'estero. Da un'indagine svolta in Italia nel 2022[1], il 46% dei campioni di miele importati dall'estero è risultato **adulterato**: ciò significa che ha subito aggiunte di acqua, sciroppo di zucchero e in certi casi anche coloranti, al fine di aumentarne la quantità riducendo i costi.

Sebbene sia difficile, dal punto di vista degli apicoltori, intervenire per contrastare le difficoltà di questo settore, c'è un comportamento ampiamente diffuso tra le api su cui si può intervenire al fine di aumentare la produzione del miele e di agevolare la riproduzione delle api: stiamo parlando della **sciamatura**.

1.2 La sciamatura delle api

La sciamatura è una **tecnica apistica** che comporta la nascita di un nuovo "organismo autonomo", ovvero un nuovo sciame. È un evento legato all'istinto di propagazione e riproduzione delle api, e solitamente si manifesta ogni anno in primavera nel periodo della grande fioritura.

Questo evento consiste nella **partenza definitiva** da un alveare di una parte delle api che lo abitano (solitamente circa dal 50% al 70%), le quali andranno a cercare una nuova casa e a costituire un nuovo sciame. In questo processo, l'ape regina fa parte del gruppo di api che lasciano l'alveare originale, al cui interno nascerà una nuova regina.

La sciamatura naturale porta alla ripopolazione degli alveari: è un grande evento, ed è attesa da molti apicoltori che sperano così di aumentare la popolazione delle api e quindi la produzione del miele. Tuttavia, spesso questo processo presenta anche alcuni svantaggi, soprattutto se accade senza che l'apicoltore se ne accorga: è consigliabile ricorrere invece alla **sciamatura artificiale**, in modo da limitare gli effetti negativi sulla produzione del miele.

1.3 I problemi per l'apicoltura

Nonostante la sciamatura sia un processo di riproduzione naturale delle api, essa è spesso considerata un problema nell'ambito dell'apicoltura a causa delle ripercussioni sia sulle api che sulla produzione di miele:

1. Durante la sciamatura, le api che lasciano l'alveare in cerca di una nuova casa potrebbero potenzialmente scomparire per sempre. È praticamente impossibile prevedere con certezza il luogo in cui il nuovo sciame andrà a stabilirsi, e di conseguenza si rischia di perdere anche più della metà delle proprie api nel caso in cui non si riesca a catturarle.

2. Siccome nel periodo antecedente alla sciamatura l'intero sciame si dedica alla riuscita dell'evento, spesso ciò si traduce in una minore produzione di miele; considerando anche che le api, prima di lasciare l'alveare, si nutrono di grandi quantità di miele per averne scorte sufficienti per i giorni successivi, è chiaro che gli apicoltori rischiano di riuscire a raccoglierne piccolissime quantità, o addirittura di non raccoglierne affatto.

3. Infine, a seguito della brusca diminuzione del numero delle api e della produzione del miele, c'è il rischio che le varie famiglie siano troppo deboli per incrementare sufficientemente la propria popolazione e produzione in tempo per superare l'inverno successivo.

1.4 Prevenzione della sciamatura

Normalmente, è possibile prevedere la sciamatura prestando attenzione ad alcuni **segnali**. Possiamo dividere questi segnali in due categorie.

1.4.1 Segnali osservabili

L'apicoltore può accorgersi autonomamente della preparazione delle api alla sciamatura controllando frequentemente l'alveare. Alcuni segnali a cui prestare particolare attenzione sono:

1. la costruzione di celle reali, le quali si trovano lungo i bordi inferiori dei favi;
2. l'elevata raccolta di polline, volta a sostenere l'aumento della popolazione;
3. l'imbiancatura della cera;
4. l'alta presenza di api ceraiole, ovvero quelle più giovani;
5. infine, a seguito dell'aumento della popolazione, la raccolta delle api in "grappoli" nelle parti inferiori degli alveari.

1.4.2 Segnali non direttamente osservabili

L'altra categoria di segnali premonitori della sciamatura è invece quella dei segnali non direttamente osservabili dall'occhio umano, ma solo attraverso l'impiego di sensori. In particolare:

1. aumento della **temperatura** interna all'alveare;
2. aumento del livello dell'**anidride carbonica** nell'aria;
3. cambiamento delle **frequenze sonore** prodotte dalle api;
4. improvvisa diminuzione del **peso** totale dell'alveare a seguito della dipartita delle api.

1.5 Le difficoltà nella prevenzione della sciamatura

In alcuni casi, prestare attenzione ai segnali osservabili può permettere di prevedere un evento sciamatorio, ma non è semplice: spesso solo gli apicoltori più esperti sono in grado di farlo.

È bene precisare che questo monitoraggio richiede l'apertura e l'**ispezione manuale** di ogni singolo alveare, anche quando questo non ha bisogno di interventi: questo è un processo invasivo, che rischia di disturbare le api, soprattutto se svolto frequentemente. Ciò può portare all'alterazione del comportamento delle api e quindi dell'equilibrio del loro sistema, senza contare che lo stress a loro causato può incidere ulteriormente sulla riproduzione della specie, che, come abbiamo visto, è già in difficoltà.

Inoltre, gli apicoltori hanno spesso molti alveari di cui occuparsi, a volte anche molto lontani tra loro, e doverli controllare uno ad uno richiede molto tempo e risorse.

Per questi motivi, negli ultimi anni si sta cercando di passare al monitoraggio dei segnali non osservabili tramite l'utilizzo di **sistemi IoT**[1] mirati all'apicoltura, autonomi e non invasivi. Questi sistemi sono però ancora poco diffusi, a causa sia dei prezzi elevati che del fatto che gli studi a riguardo sono pochi e comunque abbastanza recenti; inoltre, può essere difficile per gli apicoltori, che spesso utilizzano nel proprio lavoro tecniche tradizionali e poco tecnologiche, riporre fiducia in sistemi di questo tipo per monitorare i propri alveari.

[1] L'internet delle cose (in inglese Internet of Things, IoT) indica la rete di oggetti fisici, che hanno sensori, software e altre tecnologie integrate al fine di connettersi e scambiare dati con altri sistemi informatici.

1.6 Obiettivo di questa tesi

In considerazione delle informazioni trattate, l'obiettivo di questa tesi è quello di ideare e realizzare un **sistema di monitoraggio degli alveari** a distanza, che permetta di controllare lo stato dell'alveare attraverso la raccolta, comunicazione e visualizzazione da remoto di alcuni dati in real-time.

Il primo obiettivo di questo progetto, più a breve termine, è quello di permettere all'apicoltore, attraverso la visualizzazione dei dati relativi all'alveare, di individuare eventuali fenomeni di sciamatura, in modo da poter intervenire il prima possibile.

Inoltre, la prima versione prototipale di questo sistema si occuperà di raccogliere e memorizzare dati relativi ad un grande periodo di tempo: in questo modo, sarà possibile in futuro analizzare questi dati e procedere con la progettazione di algoritmi atti ad identificare questi eventi in maniera autonoma.

Grazie al supporto dell'azienda Quinck, ci sarà la possibilità di testare il sistema all'interno di un alveare vero e proprio, e quindi di raccogliere ed analizzare dati realistici e significativi.

Capitolo 2

Analisi

2.1 Analisi dei Requisiti

La prima parte della fase di analisi è volta a definire quali sono le funzionalità e le caratteristiche essenziali che il progetto deve raggiungere per potersi considerare completo.

1. **Visualizzazione dei dati**
   L'obiettivo principale del progetto è quello di permettere all'apicoltore di visualizzare dei dati raccolti all'interno dell'alveare, in modo da permetterne il monitoraggio da remoto.

   Sarà quindi necessario implementare un'interfaccia grafica che sia facilmente comprensibile anche da utenti poco familiari con l'informatica, mettendo a disposizione strumenti di analisi dei dati quali schemi e grafici.

2. **Memorizzazione dei dati**
   È fondamentale occuparsi della memorizzazione dei dati, poiché dovrà essere possibile consultare anche i dati raccolti in precedenza. Inoltre, memorizzare i dati permanentemente potrà servire in futuro per svolgere analisi su di essi e proseguire lo sviluppo del sistema.

3. **Alimentazione e connessione**
   I sensori dovranno essere posizionati all'interno di un alveare, lontano dalla corrente elettrica e da reti wi-fi. Sarà necessario includere nella progettazione del sistema soluzioni a questi due problemi, come per esempio l'utilizzo di una batteria e il collegamento ad internet tramite rete mobile.

4. **Resistenza all'ambiente**
   L'altra conseguenza del posizionamento dei sensori è l'esposizione sia agli agenti atmosferici (vento, pioggia e basse temperature) che alle api stesse. È necessario progettare la parte hardware in modo che sia robusta e resistente nel tempo.

2.2 Analisi delle tipologie di dati

Una seconda fase di analisi è stata svolta riguardo alle varie tipologie di dati che possono essere raccolte, per capire quali di esse sono più significative al fine di individuare un evento di sciamatura e stabilire la difficoltà nella misurazione. Sulla base dei sistemi di monitoraggio degli alveari attualmente in commercio, sono stati presi in considerazione la temperatura, il peso dell'alveare, l'umidità e le frequenze sonore.

2.2.1 Temperatura

La prima tipologia di dato presa in analisi è la temperatura. In linea generale, la temperatura interna all'alveare rimane tendenzialmente stabile, presentando variazioni molto lievi e limitate a circa 1°C; tuttavia, è stato verificato che nel periodo immediatamente precedente alla sciamatura è possibile notare un **aumento anomalo** della temperatura dovuto al riscaldamento dei muscoli necessari al volo delle api.

Durante un esperimento riguardante dieci diversi alveari svolto in Norvegia nel 2015[4], sono stati notati innalzamenti della temperatura in corrispondenza di ogni fenomeno di sciamatura osservato. La temperatura è aumentata in tutti i casi di 1.5-3.4°C, arrivando a registrare fino a 38°C, in periodo compresi tra gli 8 e i 20 minuti precedenti all'evento sciamatorio, per poi tornare ai valori originali in breve tempo.

Possiamo osservare i dati raccolti in un alveare a Jelgava, in Lettonia, il 31 maggio 2018, durante un'altra ricerca[5]: la freccia nel grafico indica la sciamatura. È evidente il brusco aumento della temperatura avvenuto poco prima di essa.

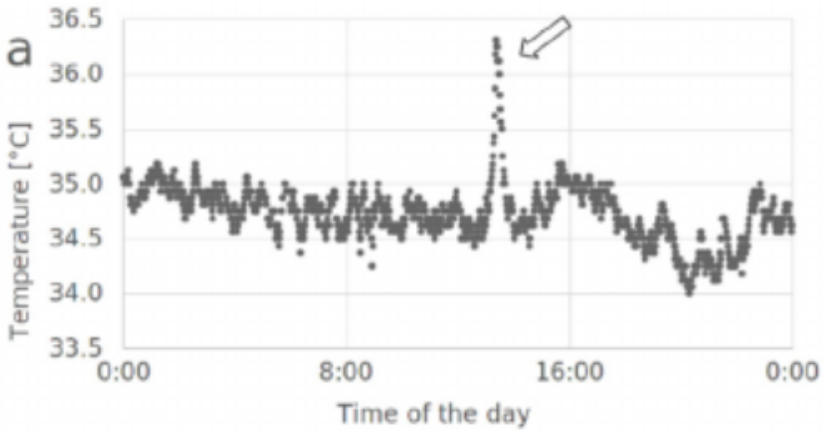

Figura 2.1: Evoluzione della temperatura all'interno di un alveare, 31 maggio 2018, Jelgava

Analizzare la temperatura si è quindi rivelato sufficiente per identificare correttamente l'imminente sciamatura, tuttavia è bene notare che l'intervallo di tempo trascorso tra l'aumento della temperatura e la dipartita delle api è decisamente non sufficiente per permettere un intervento preventivo.

L'identificazione della sciamatura può però permettere all'apicoltore di effettuare un tentativo di **recupero e riaddomesticazione** della famiglia di api prima che sia troppo tardi, in quanto esse trascorrono da qualche ora fino a qualche giorno nelle vicininanze dell'alveare originale in attesa di trovare una postazione adatta in cui stabilirsi, rimanendo solitamente appese ai rami degli alberi.

2.2.2 Peso

Come per la temperatura, anche il peso dell'arnia può essere un buon indicatore di un fenomeno di sciamatura. La dipartita di una grande quantità di api in contemporanea causa infatti, come si può immaginare, una **brusca diminuzione** del peso totale. Seppure il peso delle singole api sia trascurabile rispetto al peso dell'arnia, che nell'apicoltura è spesso una grande struttura di legno, la grande quantità di api che lasciano contemporaneamente l'alveare può risultare in diminuzioni istantanee anche dell'ordine dei chilogrammi.

Consideriamo nuovamente la precedente ricerca svolta in Lettonia[5], che ha visto anche il monitoraggio del peso di un alveare a Vecauce nel 2021: anche per quanto riguarda il peso, possiamo notare una variazione significativa in corrispondenza del'evento di sciamatura.

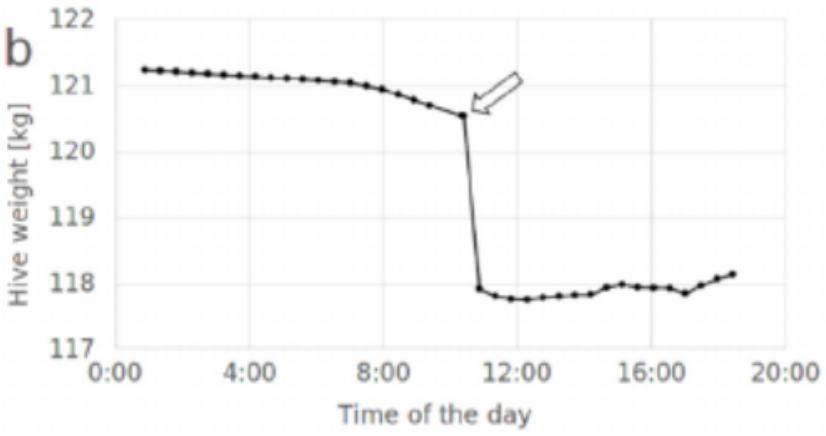

Figura 2.2: Evoluzione del peso di un alveare, 11 giugno 2021, Vecauce

Sebbene il peso dell'alveare risulti essere decisamente significativo nell'identificare la sciamatura, bisogna considerare le difficoltà causate da questa misurazione: sarebbe semplice, infatti, pesare manualmente l'alveare con una bilancia, ma molto più complicato automatizzare il trasferimento di questo dato ad un sistema IoT. Per questo motivo, purtroppo, la raccolta di dati relativi al peso dell'alveare è stata esclusa, e ci si affiderà alla temperatura per identificare gli eventi sciamatori già avvenuti.

2.2.3 Umidità

Riguardo all'umidità, le informazioni disponibili sembrano essere decisamente minori. È noto che, similmente alla temperatura, anche l'umidità rimane tendenzialmente stabile all'interno dell'alveare, con le variazioni più significative in corrispondenza, anche in questo caso, degli eventi di sciamatura. Tuttavia, queste variazioni non sembrano sufficientemente grandi per garantire la corretta identificazione di tali eventi, pertanto l'utilizzo di sensori atti a misurare l'umidità è stato escluso.

2.2.4 Frequenze sonore

Infine, l'ultimo tipo di dato da analizzare è quello delle frequenze sonore. Questa è forse la tipologia più ampiamente studiata ed utilizzata nel campo dell'apicoltura, poichè fin dall'antichità sono stati identificate alcune correlazioni tra il ronzio delle api e gli eventi sciamatori: già Aristotele parlava di un suono monotono e particolare emesso dalle api a partire da alcuni giorni prima della sciamatura.

Prendiamo in analisi uno studio[3] che nel 2014 ha raccolto dati sulle frequenze sonore relative a due diverse colonie di api, di cui nella prima è avvenuta la sciamatura e nella seconda no.

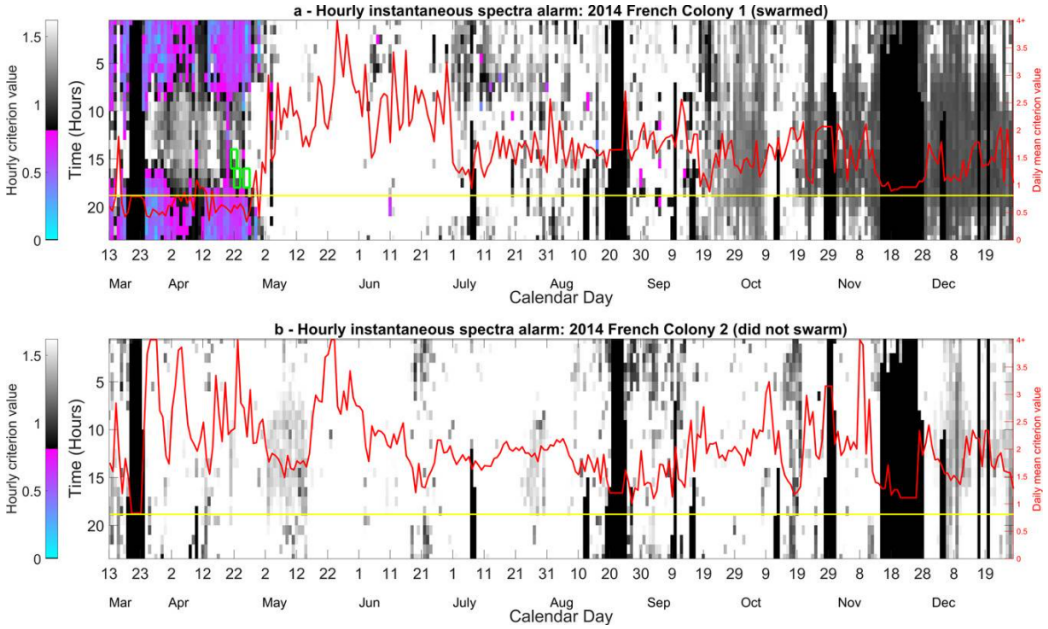

Figura 2.3: Applicazione di un algoritmo di classificazione ai valori delle frequenze sonore raccolte dal 13 marzo all'1 dicembre 2014

Possiamo vedere come sia stato possibile, applicando alle frequenze sonore determinati algoritmi di classificazione, distinguere in maniera piuttosto decisa i valori raccolti in fase di preparazione alla sciamatura (scala azzurro-viola) da quelli raccolti durante lo stato normale delle api (scala bianco-nero).

È particolarmente interessante notare che, in questo caso, è stato possibile identificare la preparazione alla sciamatura con un anticipo di **quasi due mesi**: ciò apre la possibilità di creare sistemi atti non solo a notificare all'apicoltore l'avvenuta sciamatura delle api, ma anche ad informarlo con un anticipo sufficiente per permettergli di intervenire e di limitare quindi la perdita delle api e della produzione di miele.

Sebbene sia necessario approfondire lo studio delle frequenze sonore all'interno degli alveari per poter sviluppare un algoritmo automatico di identificazione della sciamatura, è stato deciso di procedere con la raccolta e la visualizzazione da remoto di questo tipo di dato, che può comunque essere significativo per notare cambiamenti nel comportamento delle api.

2.3 Come misurare le frequenze sonore

Una volta scelte le tipologie di dati che si vogliono raccogliere, occorre individuare gli strumenti e le tecnologie necessari per farlo. La misurazione della temperatura è molto semplice: una singola lettura attraverso un sensore apposito è sufficiente per ottenere un valore significativo. Al contrario, il valore ottenuto da una singola misurazione con un microfono non fornisce, di per sé, alcuna informazione utile: è necessario comprendere meglio come funziona il suono e come esso sia rappresentato digitalmente.

2.3.1 L'audio e le onde sonore

In natura, il suono è una vibrazione meccanica che si propaga nell'aria e per altri mezzi di trasmissione. Possiamo pensare a questa vibrazione come ad un segnale continuo nel tempo, come per esempio un'onda sinusoidale[1].

Ad ogni onda sonora corrisponde una determinata **frequenza**[2] sonora, espressa in **Hz**[3]; nel caso della sinusoide, la frequenza quantifica il numero di ripetizioni dell'onda sinusoidale.

In realtà, i suoni da noi percepiti sono molto più complicati: i segnali audio presenti nel mondo reale sono più irregolari, e quello che sentiamo è composto da tanti segnali audio, e quindi tante vibrazioni, ognuna con la propria frequenza.

[1] Una sinusoide è un'onda descritta matematicamente dalla funzione seno.

[2] La frequenza è la grandezza che indica quante volte un fenomeno viene ripetuto nel tempo.

[3] Gli Hertz (Hz) sono l'unità di misura della frequenza; 1Hz equivale ad una volta al secondo.

2.3.2 L'audio digitale e il campionamento

Il suono analogico può essere campionato, attraverso l'impiego di un microfono, per misurare i valori del segnale originale in alcuni precisi istanti. Più i campionamenti vengono effettuati a intervalli brevi, più la successione dei valori campionati sarà fedele al segnale audio originale.

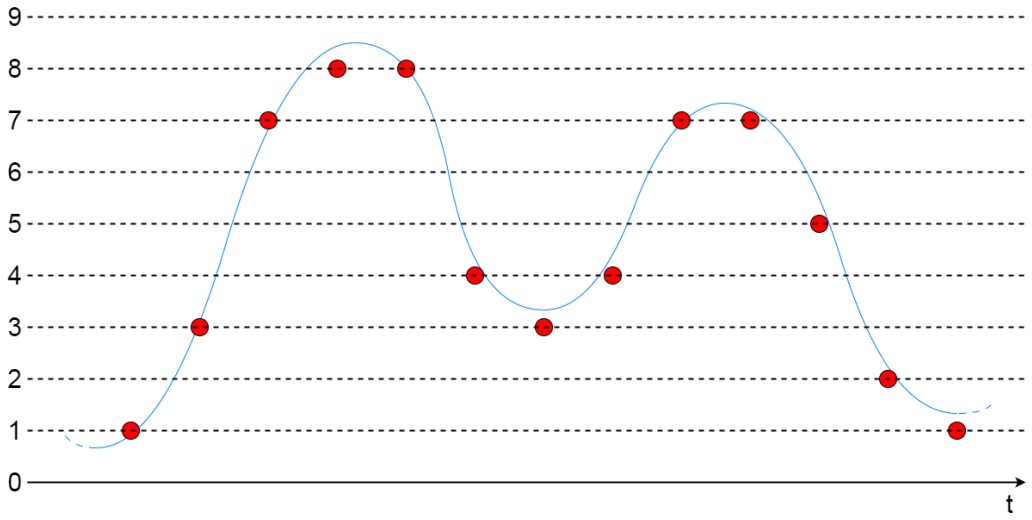

Figura 2.4: Esempio di campionamento di un segnale audio

Il suono digitale corrisponde quindi alla successione di tanti valori campionati in breve tempo.

Poichè l'audio digitale sia significativo, è necessario effettuare le varie misurazioni ad una precisa **frequenza di campionamento** (da non confondere con le frequenze sonore!), che indica il numero di campionamenti presi al secondo.

Come abbiamo visto, tuttavia, i valori da monitorare per poter prevedere un fenomeno di sciamatura sono quelli relativi alle frequenze sonore che producono un determinato suono, e non il suono di per sé; è però possibile ricavare le frequenze sonore a partire da un segnale audio digitale.

2.3.3 Analisi delle frequenze di un segnale audio

È possibile analizzare un segnale audio e risalire alle frequenze che lo compongono utilizzando la **trasformata di Fourier**. Si tratta di un potente strumento matematico che permette di analizzare l'audio da un nuovo punto di vista, passando dall'evoluzione del segnale nel tempo al valore delle varie frequenze che producono tale segnale.

Per ottenere dei risultati significativi, tuttavia, è necessario tenere in considerazione che:

- più numerosi sono i campionamenti effettuati, più frequenze sonore sarà possibile individuare;
- i campionamenti devono essere effettuati con una frequenza di campionamento ben definita.

In particolare, la frequenza di campionamento è fondamentale per poter interpretare correttamente i risultati ottenuti.

2.4 Casi d'uso

Dato che il sistema intende permettere il monitoraggio da remoto dell'alveare, ed ha quindi un funzionamento tendenzialmente passivo, i casi d'uso da parte dell'apicoltore risultano essere banalmente la lettura dei dati relativi alla temperatura e alle frequenze sonore.

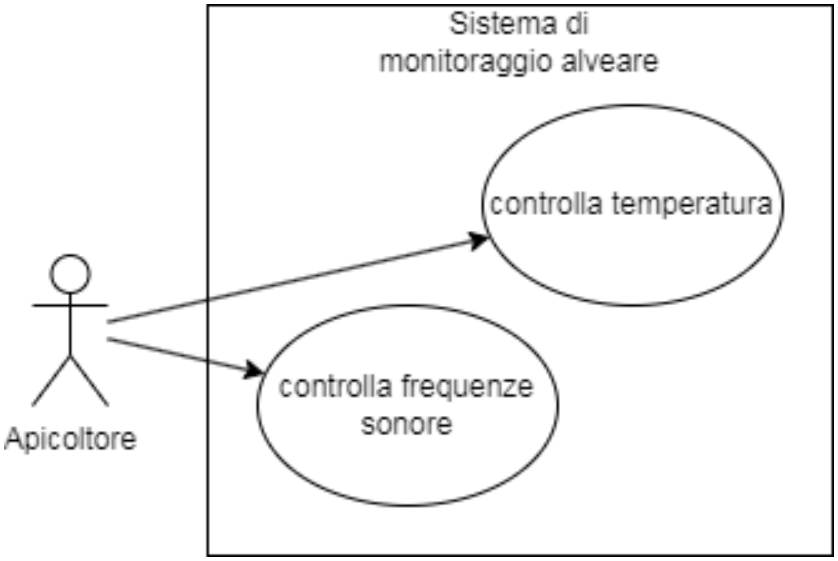

Figura 2.5: Diagramma dei casi d'uso

Capitolo 3

Progettazione

3.1 Divisione in sottosistemi

Come prima fase della progettazione del sistema, è stato necessario identificare quali sono i **sottosistemi** che lo compongono.

Siccome i dati relativi alla temperatura e alle frequenze sonore devono necessariamente essere raccolti all'interno dell'alveare, servirà sicuramente un sottosistema dedicato all'alimentazione dei sensori e alla lettura dei dati. All'interno di questo sottosistema, che chiameremo **HiveSubsystem**, servirà un microcontrollore per interagire con i sensori e comunicare i dati raccolti. Il programma che verrà eseguito dal microcontrollore e che si occuperà di ciò verrà chiamato **HiveController**.

I dati dovranno essere accessibili e consultabili attraverso un **Frontend**[1], che può essere ad esempio un programma o una pagina web. Il Frontend permetterà di visualizzare i valori attuali e passati, fornendo dei grafici per agevolare la lettura.

Sarà necessario, infine, un sottosistema **Backend**[2] atto a memorizzare i dati misurati dai sensori, in modo da poterli comunicare al Frontend. Il Backend dovrà rimanere costantemente in ascolto su due diversi canali, da un lato per ricevere periodicamente i nuovi dati ottenuti dalle misurazioni effettuate nell'HiveSubsystem, e dall'altro per fornire al Frontend tali dati.

[1] Il frontend è la parte di un programma visibile all'utente, con cui egli può interagire.
[2] Il backend è la parte del programma che rende possibile il funzionamento del frontend.

3.2 Comunicazione tra i sottosistemi

In sistemi di questo tipo i due protocolli di comunicazione più diffusi sono **HTTP** e **MQTT**. Anche se a livello pratico la situazione è più complicata, possiamo considerare l'HiveSubsystem e il Frontend come i due sottosistemi da cui "partono" le connessioni, e il Backend come quello che le "riceve".

3.2.1 Connessione HiveSubsystem-Backend

La prima connessione del sistema serve per comunicare i dati misurati dai sensori all'interno dell'HiveSubsystem al Backend. Il protocollo scelto per realizzare questa connessione è MQTT: questo protocollo di tipo **publish-subscribe** è infatti più leggero ed è stato progettato appositamente per le situazioni in cui è richiesto un basso impatto energetico.

Il protocollo MQTT fa uso del concetto dei **topic**, ovvero argomenti di messaggi, e prevede la presenza di tre partecipanti alla conversazione, chiamati **publisher**, **subscriber** e **broker**. Quest'ultimo in particolare ha il ruolo di ridistribuire i messaggi pubblicati.

In sintesi, i subscriber si connettono al broker e si "iscrivono" ad uno o più determinati topic. Quando un publisher vuole pubblicare dei dati relativi ad un determinato topic, lo fa inviandoli al broker: esso si occuperà quindi di comunicare i dati pubblicati a tutti i subscriber iscritti a tale topic.

3.2.2 Connessione Frontend-Backend

La seconda connessione presente nel sistema è quella che permette la comunicazione dei dati memorizzati nel Backend al Frontend, per poi poter essere visualizzati.

Questa connessione verrà invece realizzata con il protocollo HTTP. Infatti, non c'è in questo caso il problema del consumo energetico, ed è più comodo utilizzare un protocollo di tipo **request-response** in cui il Frontend richiede i dati solo quando essi devono essere visualizzati, e il Backend inserisce i dati richiesti all'interno della risposta.

3.3 HiveSubsystem

Passiamo ora alla progettazione effettiva dell'HiveSubsystem. Per quanto riguarda l'hardware, sarà necessario definire quali componenti hardware verranno utilizzati, in che modo saranno collegati tra di loro e come dovrà comportarsi questo sotto-sistema.

3.3.1 Componenti del circuito

Microcontrollore e connessione a internet

Il microcontrollore inizialmente scelto per alimentare e far funzionare i sensori è un **esp32**, invece di altri più comuni come Arduino o Raspberry, a causa del ridotto consumo elettrico. Questo sistema dovrà infatti essere collocato in prossimità dell'alveare e, di conseguenza, lontano da fonti di alimentazione.

I microcontrollori esp32, oltre ad avere consumi energetici molto bassi, dispongono di funzionalità di risparmio energetico particolarmente utili, che potranno essere utilizzate durante le fasi di "riposo" dell'HiveSubsystem.

Successivamente, dato che nelle vicinanze dell'alveare non sono presenti connessioni wi-fi, si è proseguito con la ricerca di un componente che permettesse al microcontrollore di connettersi alla rete mobile. Il modulo che permette la connessione ad internet attraverso la rete mobile è chiamato **SIM800L**, e per il proprio funzionamento utilizza la connessione alla rete 2G attraverso l'impiego di una scheda SIM.

Nell'interfacciare il modulo SIM800L al microcontrollore esp32 emerge però un problema: per alimentare il modulo SIM800L occorre una tensione compresa tra 3.4V e 4.4V, e l'esp32 lavora con tensioni fino a 3.3V. Non sarebbe quindi possibile utilizzare questi due componenti insieme, se non affiancando al microcontrollore una batteria esterna in grado di fornire al modulo la tensione necessaria al corretto funzionamento. Sebbene questa sia una possibilità, la necessità di un'ulteriore batteria e l'ulteriore complessità del circuito rappresentano dei nuovi ostacoli.

È stata poi individuata una scheda particolarmente adatta a questo progetto, che risolve questi problemi: **TTGO T-Call esp32 SIM800L**. Si tratta di un modulo compatto che integra autonomamente le funzionalità del SIM800L ad un microcontrollore esp32, eliminando il problema dell'alimentazione. Per realizzare il sistema sarà quindi utilizzato questo microcontrollore.

Sensore di temperatura

Per quanto riguarda la misurazione della temperatura, è stato scelto di utilizzare un sensore **DS18B20**, con un raggio di misurazione da -55°C a 125°C, ampiamente sufficiente, ed un'elevata precisione.

Il sensore dispone di tre pin, dei quali due devono essere collegati all'alimentazione (**VCC**) e a massa (**GND**). Il terzo pin, è invece quello da cui si legge il valore risultante, che corrisponde alla temperatura in gradi centigradi.

Microfono

Per provvedere al campionamento dei suoni interni all'alveare sarà utilizzato un microfono **MAX9814**, di alta qualità e provvisto di amplificatore.

Questo microfono dispone di cinque pin, tra cui quelli di alimentazione (**VCC**), massa (**GND**) e output (**OUT**). Gli ultimi due pin disponibili, **AR** e **GAIN**, sono opzionali e servono per controllare rispettivamente attack/release ratio e amplificazione.

Alimentazione

Come fonte di alimentazione sarà inizialmente utilizzato, come soluzione temporanea, un **power-bank** da **10.000mAh**[3].

In fase di sviluppo sarà svolta un'analisi relativa al consumo energetico del circuito, per valutare poi l'eventuale sostituzione con altri tipi di batterie, eventualmente con ricarica a pannello solare.

Varie

Oltre ai componenti già trattati verranno utilizzati:

- una **breadboard**[4];
- vari **connettori**;
- una **microSIM** per la connessione alla rete mobile;
- una **resistenza da 4.7kΩ** necessaria per il sensore di temperatura;
- un **cavo USB/USB-C** per collegare il chip al pc o al power-bank.

[3]milli-Ampere-ora: ad esempio, con un consumo ipotetico di 100mA una batteria da 10.000mAh durerebbe 10.000/100 = 100 ore.

[4]La breadboard è uno strumento utilizzato per creare prototipi di circuiti elettrici senza il bisogno di saldature.

3.3.2 Circuito

Di seguito lo schema del circuito utilizzato durante lo sviluppo del progetto.

In fase di programmazione, collegando il chip esp32 ad un pc attraverso la porta USB-C si potrà scrivere il programma sul microcontrollore, mantenendo al contempo il circuito alimentato.

Quando il sistema verrà posizionato all'interno dell'alveare, invece, il chip sarà direttamente collegato al power-bank, la breadboard sarà rimossa, e verranno utilizzati cavi coperti e resistenti alle condizioni atmosferiche.

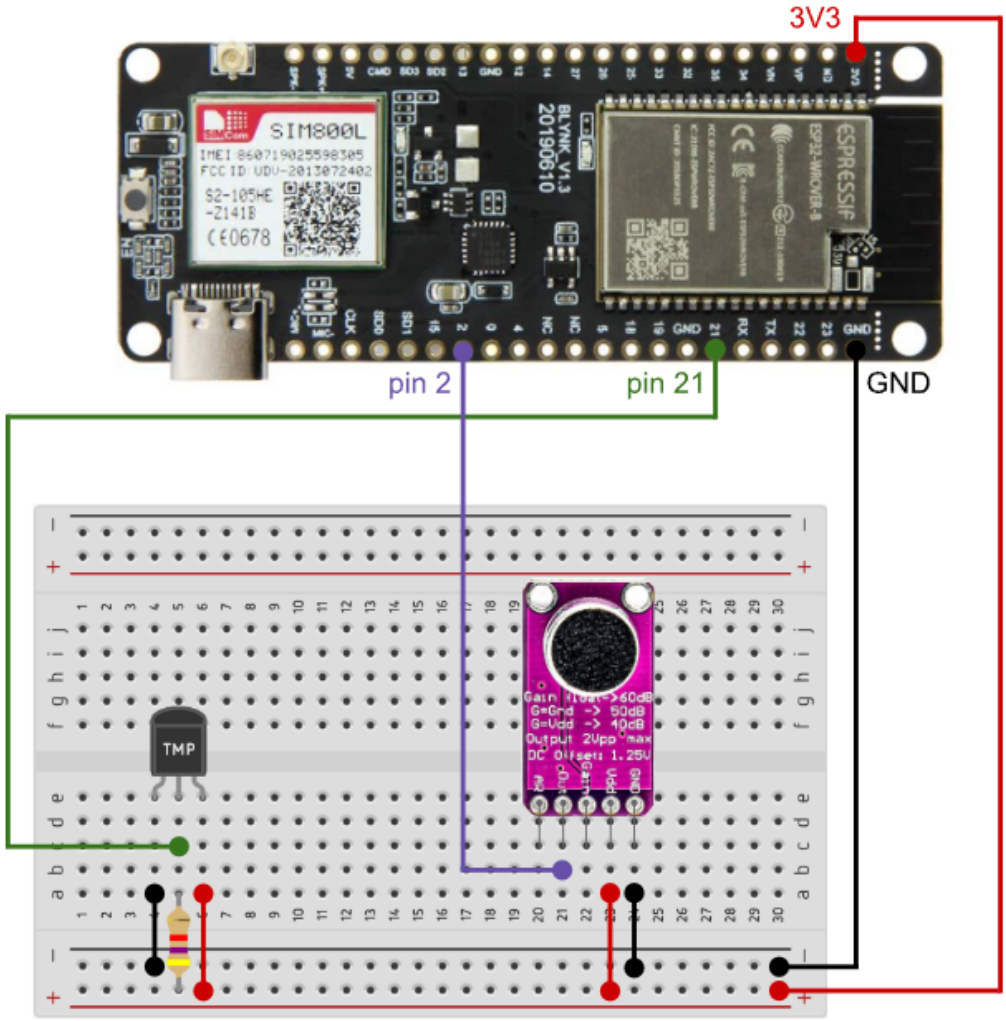

Figura 3.1: Schema del circuito dell'hive sub-system

3.3.3 Software

Il microcontrollore dovrà principalmente occuparsi di raccogliere ed inviare i dati, azioni la cui esecuzione è molto rapida. Inoltre, non è necessario raccogliere i dati in maniera continua, anche perché in questo modo la quantità di dati da visualizzare sarebbe esageratamente ed inutilmente elevata: è sufficiente eseguire la misurazione ad intervalli di tempo regolari.

Per questo motivo, il programma sarà composto da due fasi che si alterneranno reciprocamente: una **fase di esecuzione** e una **fase di risparmio energetico**.

- Durante la **fase di esecuzione**, il microcontrollore si occuperà prima di tutto di instaurare la connessione ad internet attraverso il modulo SIM800L integrato.

  Una volta effettuata la connessione, si potrà procedere con la misurazione della temperatura e con il campionamento dell'audio, per poi trasmettere i dati raccolti al Backend.

- Dopo aver terminato l'invio dei dati, si passerà alla **fase di risparmio energetico**. Questa fase consente al microcontrollore di ridurre al minimo il consumo elettrico, cosa particolarmente utile per questo progetto data la mancanza di una fonte di alimentazione stabile, e quindi la necessità di usare una batteria. Trascorso un intervallo di tempo prefissato, il programma tornerà alla fase di esecuzione e ricomincerà dall'inizio.

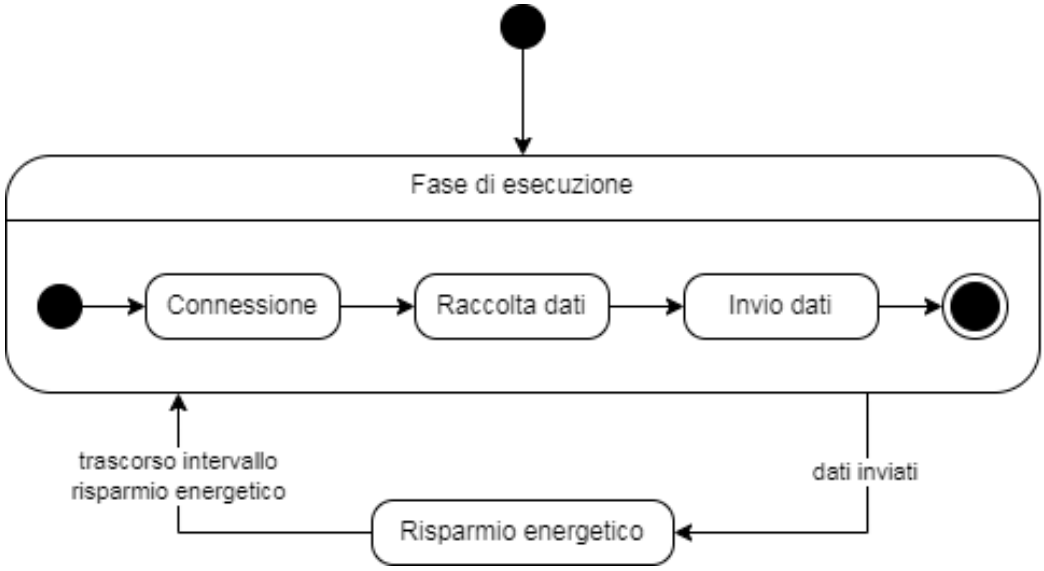

Figura 3.2: Diagramma degli stati del modulo dei sensori

3.3.4 Durata della fase di risparmio energetico

La durata della fase di risparmio energetico deve essere scelta accuratamente: un periodo di pausa troppo prolungato rischierebbe di non permettere di effettuare misurazioni sufficienti per identificare un evento di sciamatura, mentre uno troppo breve costringerebbe il sistema ad effettuare il riavvio dei componenti e della connessione molto frequentemente, incidendo così sul consumo elettrico.

Riprendendo in analisi lo studio relativo al'identificazione della sciamatura attraverso la temperatura[4], possiamo considerare un periodo di riscaldamento delle api precedente alla sciamatura compreso normalmente tra gli 8 e i 20 minuti. In questo intervallo di tempo occorre effettuare almeno una misurazione, ma è comunque meglio effettuarne di più per migliorare l'affidabilità: nello studio sopracitato le misurazioni sono state effettuate una volta al minuto. Nell'analisi relativa alle frequenze sonore[3], le misurazioni sono state effettuate invece ogni 3 minuti.

L'intervallo di risparmio energetico scelto inizialmente è di **2 minuti**. Durante l'implementazione, saranno svolti dei test sia per quanto riguarda il consumo energetico che la raccolta dei dati, e si potrà quindi andare a modificare la durata in base alle esigenze.

3.4 Backend

Abbiamo già visto che il sottosistema Backend avrà il compito di ricevere e memorizzare le misurazioni effettuate dai sensori, per poi inviarle al Frontend ogni volta che sono richieste. Il funzionamento del Backend è dato da due programmi che chiameremo **DataReceiver** e **DataProvider**.

3.4.1 DataReceiver e DataProvider

Il **DataReceiver** è il programma che si occupa di ricevere i dati raccolti e inviati dal modulo dei sensori e di memorizzarli permanentemente, fungendo da subscriber nella connessione MQTT; ha anche il compito di implementare la trasformata di Fourier per l'analisi delle frequenze audio, che richiede calcoli troppo dispendiosi in termini energetici per l'HiveController.

Il **DataProvider**, invece, si occupa di recuperare i dati memorizzati e fornirli al Frontend ogni volta che sono richiesti. Questi due programmi costituiscono il cuore del sottosistema Backend e ne definiscono il comportamento.

La scelta di separare questi due programmi è data principalmente dal fatto che se in futuro dovessero essere effettuate delle modifiche ad uno dei due (per esempio l'aggiunta di ulteriori filtri alla richiesta di dati al DataProvider), l'altro potrebbe continuare normalmente la sua esecuzione, riducendo così la possibilità di una perdita di dati.

Per la realizzazione di questi programmi è stato deciso di utilizzare **Javascript** e **Node.js**, tecnologie già ampiamente diffuse nel campo della programmazione server.

3.4.2 Database

Il **Database** è il luogo in cui verranno salvati i dati, per poi poter essere recuperati in qualunque momento e inviati al Frontend. Per questo progetto sarà utilizzato **MongoDB**, un database di tipo **documentale**. In questo modo, i dati saranno salvati in un formato direttamente compatibile con **JSON**, e saranno quindi già pronti per essere comunicati al Frontend senza bisogno di essere modificati: JSON può infatti essere utilizzato come formato di trasmissione dei dati all'interno delle comunicazioni HTTP.

3.4.3 Broker

Abbiamo visto che per instaurare una connessione con il protocollo MQTT è necessario un Broker per la gestione dei messaggi. Il Broker utilizzato è **Mosquitto**, e farà anch'esso parte del sottosistema Backend.

3.4.4 Containerizzazione

Una volta definiti quali sono i componenti del Backend, occorre anche decidere dove e come questi programmi verranno eseguiti. Per facilitare l'esecuzione e la comunicazione dei vari componenti del Backend si è deciso di ricorrere alla **containerizzazione**, racchiudendoli ognuno nel proprio **container**.

I container sono simili a piccole macchine virtuali, la cui unica funzione è quella di eseguire il programma per cui sono state create. Tra gli altri vantaggi hanno anche quello di eliminare i problemi relativi alla portabilità del software, poiché ogni container funziona allo stesso modo a prescindere dalla macchina su cui viene eseguito.

I container sono creati ed eseguiti a partire dalle **immagini**: esse possono essere personalizzate, e quindi create a partire dalle proprie applicazioni, ma esistono anche immagini pubbliche che possono essere utilizzate da chiunque. Per il Backend verranno utilizzate due immagini personalizzate Node.js per il DataReceiver e il DataProvider, più quella di MongoDB per il Database e quella di Mosquitto per il Broker MQTT.

Grazie alla containerizzazione sarà possibile, in fase di sviluppo, testare il sistema in locale: i container verranno eseguiti e comunicheranno fra loro come se si trovassero già online.

3.5 Frontend

Lo scopo del sottosistema Frontend sarà quello di visualizzare i dati memorizzati all'interno del Database, e si ha quindi molta libertà di scelta per quanto riguarda le tecnologie utilizzate per realizzarlo: ad esempio, si potrebbe realizzare un'app per smartphone o un programma per pc.

3.5.1 Tecnologie utilizzate

Per rendere i dati facilmente consultabili ovunque la scelta è ricaduta sulla realizzazione di una pagina web, in modo da poter essere utilizzata su qualunque tipo di dispositivo. In questa pagina saranno presenti dei grafici attraverso cui poter consultare l'andamento della temperatura nel tempo e lo spettro delle frequenze sonore, con la possibilità di scegliere l'intervallo di tempo di riferimento.

La struttura della pagina sarà realizzata in **HTML**, e ci sarà uno script **Javascript** integrato nella pagina che permetterà di recuperare i dati dal Backend tramite l'impiego di richieste HTTP asincrone, oltre a gestire la visualizzazione degli stessi attraverso i grafici.

3.6 Diagramma dei componenti

Ora che abbiamo definito tutti i componenti e sottocomponenti che fanno parte del sistema, possiamo rappresentarne la struttura attraverso un diagramma UML.

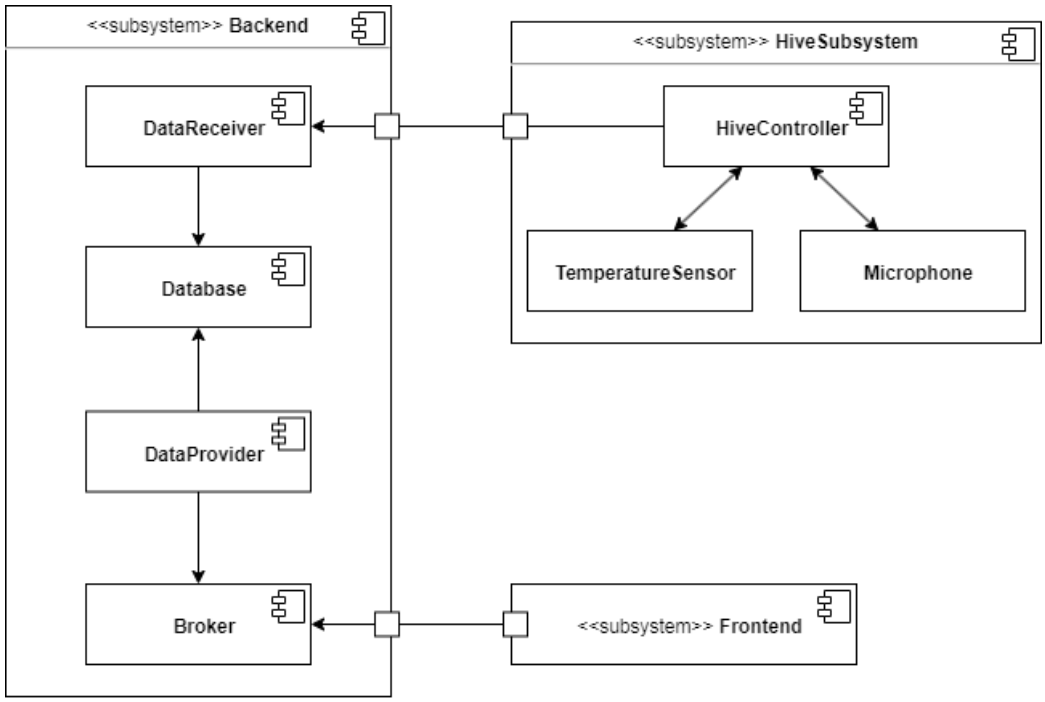

Figura 3.3: Diagramma UML dei componenti del sistema

Capitolo 4

Sviluppo e implementazione prototipale

4.1 HiveSubsystem

Il programma che verrà eseguito dal microcontrollore è scritto all'interno di un file con estensione `.ino`. Il linguaggio di programmazione è basato su `C++`, con l'aggiunta di alcuni comandi e definizioni supplementari.

4.1.1 Codice dell'HiveController

Raccolta dei dati

Mentre la misurazione della temperatura è immediata, è più interessante analizzare il campionamento dell'audio.

```
#define FREQ 10
#define SAMPLES 1024

sampling_period_us = round(1000ul * (1.0 / MAX_FREQ));

for (int i = 0; i < SAMPLES; i++) {
  unsigned long chrono = micros();
  data[i] = analogRead(MIC);
  while (micros() - chrono < sampling_period_us);
}
```

Le due costanti `FREQ` e `SAMPLES` indicano rispettivamente la frequenza di campionamento (in KHz) e il numero di campionamenti da effettuare. La variabile `sampling_period_us` indica invece il periodo di tempo che deve trascorrere tra un campionamento e quello successivo, espresso in microsecondi: è stato usato un ciclo `while` vuoto per far trascorrere questo tempo.

Invio dei dati

```
#define SAMPLES 1024
#define SAMPLES_PER_MESSAGE 128
#define MSG_BUFFER_SIZE 1024

char msg[MSG_BUFFER_SIZE];

int messages = SAMPLES/SAMPLES_PER_MESSAGE;
for (int i = 0; i < messages; i++) {
  String audioString = "{\"messages\": " + String(messages) +
    ",\"sequence_number\": " + String(i) + ", \"values\": [";
  for (int l = 0; l < SAMPLES_PER_MESSAGE; l++) {
    audioString = audioString + String(data[i *
   SAMPLES_PER_MESSAGE + l]);
    if (l < SAMPLES_PER_MESSAGE - 1) {
      audioString = audioString + ", ";
    }
  }
  audioString = audioString + "]}";
  audioString.toCharArray(msg, MSG_BUFFER_SIZE);

  mqtt.publish(audioTopic, msg);
  mqtt.loop();
}
```

I valori campionati dal microfono vengono inseriti in una stringa in formato JSON: in questo modo il Backend potrà processarli direttamente. Alcune particolarità da notare:

- In fase di sviluppo è stato notato che messaggi particolarmente lunghi causano problemi nella trasmissione: per questo motivo, è stato fatto in modo di spezzare il campionamento in più messaggi.

- All'interno di ogni messaggio il valore `messages` indica il numero totale di messaggi inviati per comunicare il campionamento, mentre `sequence_number` serve a ricostruire l'ordine corretto dei messaggi nel caso l'ordine di ricezione non sia quello giusto.

- Siccome la funzione `mqtt.publish()` richiede che il messaggio da pubblicare sia un array di `char` e non una stringa, è necessario trascrivere i messaggi all'interno del buffer `msg`.

4.1.2 Modalità di consumo energetico

I microcontrollori esp32 dispongono di cinque diverse modalità di consumo energetico.

1. **Active**
   La modalità principale di operatività di esp32: tutti i componenti sono attivi, e si ha il consumo energetico maggiore, solitamente oltre ai 100mA, raggiungendo picchi anche fino ai 700/800mA nel caso di utilizzo contemporaneo di wi-fi e bluetooth.

2. **Modem-sleep**
   Simile alla modalità attiva, ma le connessioni wi-fi e bluetooth sono disattivate; in base all'utilizzo della CPU si hanno consumi tra i 3 e i 50mA.

3. **Light-sleep**
   In questa modalità, la CPU viene messa in pausa per ridurre il consumo energetico, e di conseguenza l'esecuzione viene interrotta; il consumo è di circa 0.8mA.

4. **Deep-sleep**
   Nella modalità deep-sleep la CPU principale viene completamente disattivata e l'esecuzione interrotta, per poi dover ricominciare dall'inizio una volta terminata questa fase; anche la memoria principale del chip viene spenta, con conseguente perdita dei dati memorizzati, e il consumo è attorno ai 10µA.

5. **Hibernation**
   Modalità molto simile alla deep-sleep, ma anche il co-processore viene disattivato, ottenendo un consumo di 2.5µA.

Le due modalità attiva e modem-sleep vengono gestite automaticamente da esp32 in base all'utilizzo o meno delle connessioni wi-fi e bluetooth. Siccome queste connessioni non sono utilizzate per questo progetto, la modalità operativa utilizzata nella fase di esecuzione sarà modem-sleep.

Allo stesso modo, anche la scelta tra la modalità deep-sleep e quella di hibernation è gestita in maniera autonoma dal microcontrollore a seguito del comando `esp_deep_sleep_start()`, utilizzando la prima solo nel caso in cui sia necessario utilizzare la memoria secondaria o uscire dal risparmio energetico a seguito di un segnale rilevato da un sensore. Di conseguenza, nella fase di risparmio energetico di questo progetto verrà utilizzata la modalità hibernation.

4.1.3 Librerie

All'interno del programma è stato fatto uso di alcune librerie, che semplificano passaggi quali la connessione ad internet e la misurazione della temperatura.

- **TinyGSM**[1]:
  Questa libreria offre alcune funzionalità che permettono di gestire il modulo SIM800L, in questo caso integrato nel microprocessore; è stata utilizzata per instaurare la connessione ad internet attraverso la rete 2G.

- **PubSubClient**[2]
  Altra libreria necessaria per la comunicazione dei dati, permette di effettuare la connessione ad un broker MQTT sia come subscriber che come publisher.

- **Dallas Temperature**[3]:
  Una libreria semplice e pratica che facilita la lettura della temperatura dal sensore, prendendo in input il numero del pin a cui è collegato il sensore e fornendo il valore letto direttamente in gradi centigradi. Per il suo funzionamento è necessaria anche la libreria **OneWire**[4]

[1] `https://github.com/vshymanskyy/TinyGSM`
[2] `https://github.com/knolleary/pubsubclient`
[3] `https://github.com/milesburton/Arduino-Temperature-Control-Library`
[4] `https://github.com/PaulStoffregen/OneWire`

4.2 Backend

Come abbiamo visto, sia il DataReceiver che il DataProvider sono stati realizzati come applicazioni Node.js: si tratta di un ambiente di runtime javascript open-source e cross-platform, ed è ormai ampiamente utilizzato nell'ambito della programmazione server.

4.2.1 Codice del DataReceiver

Anche in questo caso, è più interessante vedere parti di codice relative alla gestione dei dati audio, più complessa rispetto a quella della temperatura.

Connessione al broker MQTT

```
// Connection to MQTT server
console.log("Connecting to MQTT server...")
const mqttClient = mqtt.connect("mqtt://" + mqttServer + ":"
    + mqttPort, {
     clean: true,
     connectTimeout: 4000,
     username: '',
     password: '',
     reconnectPeriod: 1000,
})

// Subscription to MQTT topics
mqttClient.on("connect", () => {
     console.log("MQTT server connected")
     mqttClient.subscribe(temperatureTopic, () => {
         console.log("Subscribed to topic " + tempTopic)
     })
     mqttClient.subscribe(audioTopic, () => {
         console.log("Subscribed to topic " + audioTopic)
     })
})

// Receiving MQTT messages
mqttClient.on("message", (topic, payload) => {
  switch(topic) {
      case (temperatureTopic):
          saveTemperature(payload.toString())
          break
      case (audioTopic):
          processAudioData(payload.toString())
          break
  }
})
```

Ricezione dei dati audio

```
const sampleRate = 10000
let lastTime = null
let audioData = null

async function processAudioData(payload) {
  let messageData = JSON.parse(payload)

  if (new Date().getTime() - lastTime > 60000 || lastTime == null) {
    lastTime = new Date().getTime()
    audioData = Array(messageData.values.length * messageData.messages).fill(-1)
  }
  for (let i = 0; i < messageData.values.length; i++) {
    audioData[i + (messageData.sequence_number * messageData.values.length)] = (messageData.values[i] / 2048 - 1)
  }
  let check = true
  for (let i = 0; i < audioData.length; i++) {
    if (audioData[i] == -1) {
      check = false
    }
  }

  if (check) {
    const spectrum = ft(audioData)
    const labeledSpectrum = spectrum.map((v, i) => {
      return {
        frequency: (sampleRate / audioData.length) * (i + 1),
        value: v
      }
    }).filter(el => {
      return el.frequency >= 20 && el.frequency <= 1000
    })

    saveSpectrum(labeledSpectrum)
  }
}
```

Ricezione e ricostruzione dei messaggi contenenti il campionamento audio. Da notare, la trasformazione dei valori campionati, che vanno da 0 a 4095, in valori decimali nel range `[-1, 1]` compatibili con la funzione `ft()` che effettua la trasformata di Fourier, e il filtraggio per salvare solo le frequenze nel range `[20, 1000]KHz`, quelle significative in questo contesto.

Salvataggio delle frequenze sonore nel Database

```
async function saveSpectrum(spectrum) {
  await client.connect()
  console.log('Connected successfully to server')
  const db = client.db(dbName)
  const collection = db.collection('audio-data')

  const date = new Date()
  const insertResult = await collection.insertOne({
    year: date.getFullYear(),
    month: date.getMonth(),
    day: date.getDate(),
    hours: date.getHours(),
    minutes: date.getMinutes(),
    values: spectrum
  })
  console.log('Inserted value =>', insertResult)
}
```

Il programma si occupa anche di provvedere a fornire data e orario per ogni misurazione effettuata: sarebbe infatti più complicato relegare questo compito all'HiveController, poiché i microcontrollori come gli esp32 non hanno conoscenza dell'orario attuale, che dovrebbe quindi essere ottenuto da fonti esterne.

4.2.2 Codice del DataProvider

Ricezione delle richieste HTTP

```
const express = require('express')
const app = express()
app.use(express.json({ limit: "1mb" }))

app.get('/frequency_values', (req, res) => {
  const year = parseInt(req.query.year)
  const month = parseInt(req.query.month)
  const day = parseInt(req.query.day)
  const start_hour = parseInt(req.query.start_hour)
  const end_hour = parseInt(req.query.end_hour)
  getFrequencyValues(year, month, day, start_hour, end_hour).
   then((result) => {
    res.setHeader('Content-Type', 'application/json')
    res.setHeader("Access-Control-Allow-Origin", "*")
    res.writeHead(200)
    res.end(JSON.stringify(result))
  })
})

app.get('/current_temperature', (req, res) => {
  ...
})

app.get('/temperatures_by_date', (req, res) => {
  ...
})

app.listen(port, () => {
  console.log(`Data Loader listening on port ${port}`)
})
```

Codice che permette di ricevere e rispondere alle richieste HTTP inviate dal Frontend. Per filtrare i dati da recuperare dal Database vengono utilizzati i **query parameters** del protocollo HTTP. All'interno della risposta HTTP i dati sono codificati in JSON.

Recupero dei dati dal Database

```
async function getFrequencyValues(year, month, day,
    start_hour, end_hour) {
  await client.connect()
  console.log('Connected successfully to server')
  const db = client.db(dbName)
  const collection = db.collection('audio-data')

  const findResult = await collection.find({
    "year": year,
    "month": month,
    "day": day,
    "hours" : { $gte :  start_hour, $lt : end_hour}
  }).toArray(function (err, result) {
    if (err) throw err
    console.log(result)
  })
  let spectrum = []
  if (findResult.length > 0) {
    for (let i = 0; i < findResult.length; i++) {
      for (let l = 0; l < spectrum.length; l++) {
        spectrum[l].value += findResult[i].values[l].value
      }
    }
    for (let i = 0; i < spectrum.length; i++) {
      spectrum[i].value /= spectrum.length
    }
  }
  return spectrum
}
```

Dopo aver effettuato la connessione al Database, vengono recuperati i dati relativi all'intervallo di tempo selezionato. Nel caso delle frequenze sonore, vengono comunicati al Frontend i valori medi nell'intervallo scelto: i dati estratti dal Database vengono progressivamente sommati nel vettore `spectrum`, per poi effettuare la divisione per il numero di misurazioni.

4.2.3 Librerie utilizzate

- **Express**[5]
  Libreria che permette di ricevere e processare richieste HTTP, utilizzata dal DataProvider.

- **MQTT**[6]
  Questa libreria permette al DataReceiver di connettersi al broker MQTT e di sottosciversi ai topic per la ricezione dei dati.

- **MongoDB**[7]
  Utilizzata sia dal DataReceiver che dal DataProvider per effettuare la connessione al Database e gestire il salvataggio e la lettura dei dati.

- **Fourier-Transform**[8]
  È stato fatto uso di questa libreria all'interno del DataReceiver per permettere l'analisi delle frequenze sonore contenute nei campionamenti audio raccolti dal microfono.

4.2.4 API del DataProvider

Il DataProvider riceve richieste HTTP su tre diversi path, in base alla richiesta da effettuare.

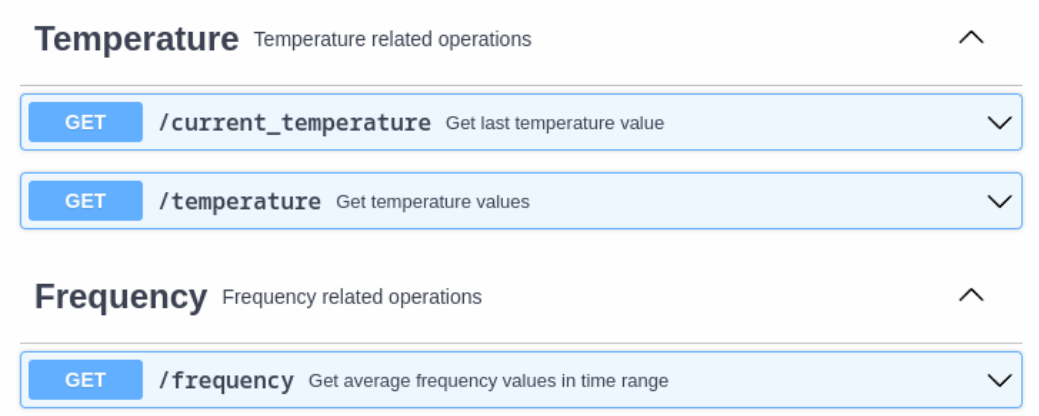

Figura 4.1: I tre tipi di richieste HTTP al data loader

[5] https://github.com/expressjs/express
[6] https://github.com/mqttjs/MQTT.js
[7] https://github.com/mongodb/node-mongodb-native
[8] https://github.com/scijs/fourier-transform

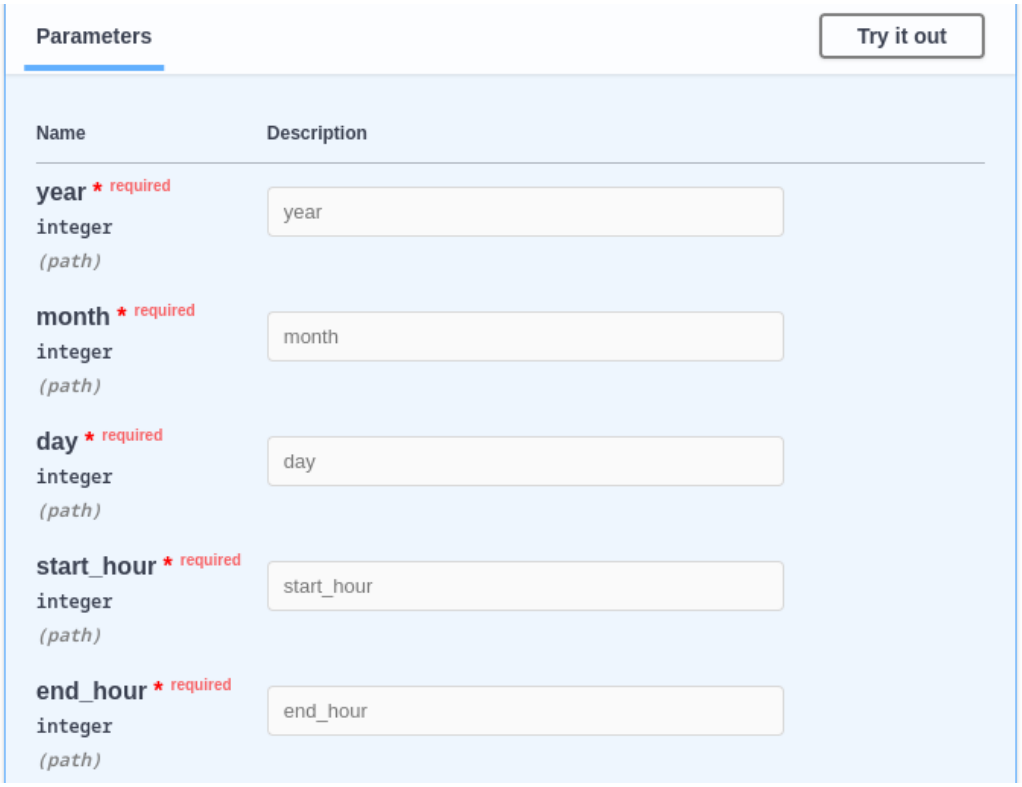

Figura 4.2: Parametri per filtrare la query nel path

Le richieste relative ai dati di temperatura e frequenze sonore prevedono l'inserimento di alcuni parametri nel path per permettere di filtrare la query in base alla data e all'orario.

Figura 4.3: Formati dei dati incapsulati nelle risposte HTTP

Per ogni dato relativo alla temperatura vengono comunicati anche la data e l'orario di misurazione; per le frequenze sonore ciò non è necessario, poiché i dati inviati sono i valori medi nell'intervallo selezionato.

4.2.5 Deployment

Il Backend deve essere sempre attivo e pronto a comunicare in qualunque momento, perciò occorre che i suoi componenti vengano eseguiti su una macchina che possa rimanere sempre attiva; inoltre è necessario che essa abbia un **indirizzo IP pubblico**, in modo che gli altri due sottosistemi possano contattarla dall'esterno.

Per soddisfare questi requisiti, è stato scelto di utilizzare una macchina virtuale di **AWS** (Amazon Web Services). AWS è un'azienda del gruppo **Amazon** che offre numerosi servizi di cloud-computing, dalle infrastrutture per il calcolo, l'archiviazione e i database fino alle nuove tecnologie, quali machine learning, intelligenza artificiale, data lake e Internet of Things.

In particolare, è stata utilizzata una macchina virtuale **EC2** (Elastic Compute Cloud) di AWS: si tratta di macchine di calcolo virtuali sicure ed estremamente versatili, ridimensionabili in base al carico di lavoro. Per lavorare con questa macchina è stata utilizzata una connessione SSH dal terminale del computer, in modo da poter comandare la macchina virtuale da remoto.

4.2.6 Docker

Per poter effettuare la containerizzazione del sistema è necessario utilizzare software appositi che si occupino di creare i container sulla base delle immagini, ed eventualmente di creare le immagini stesse a partire dalle applicazioni. Per questo progetto è stato utilizzato **Docker**, ed in particolare la sua funzionalità **Compose**.

Una volta definite le immagini che Docker utilizzerà, la funzionalità Compose permette di creare ed eseguire i container come un unico sistema. È possibile definire volumi di memorizzazione e reti di comunicazione tra i vari container, oltre a decidere per ogni container quali porte esporre agli altri container e/o verso l'esterno.

Sulla macchina virtuale EC2 è stato quindi installato Docker e, dopo aver fornito le immagini necessarie, è stato utilizzata la funzionalità Compose per eseguire tutte le applicazioni del Backend come se fossero un sistema unico.

Nel file `docker-compose.yml` è indicata la struttura del sistema, e oltre ai container da creare e alle immagini da utilizzare sono definiti:

- le reti virtuali da creare (una tra DataReceiver e Database, una tra Database e DataProvider e una tra Broker e DataReceiver);
- i volumi di memorizzazione da creare (per la memorizzazione del Database);
- le porte da esporre verso l'esterno (una per il DataReceiver e una per il DataProvider).

File docker-compose.yml

```
version: '3.8'
services:
  # DATA RECEIVER
  data-receiver:
    build: ./data-receiver
    image: data-receiver-image
    container_name: data-receiver
    environment:
      - MONGODB_URL=mongodb://mongo-db:27017/mongo_db
    depends_on:
      - mongo-db
    networks:
      - data-receiver-network
      - mqtt-network
    restart: on-failure

  # DATA PROVIDER
  ...

  # DATABASE
  mongo-db:
    image: mongo:latest
    container_name: mongo-db
    hostname: mongo-db
    command: mongod --port 27017
    volumes:
      - mongo-db:/data/db
    networks:
      - data-receiver-network
      - data-loader-network

  # MQTT BROKER
  ...

volumes:
  mongo-db:

networks:
  data-receiver-network:
    driver: bridge
  data-provider-network:
    driver: bridge
  mqtt-network:
    driver: bridge
```

4.3 Frontend

La struttura della pagina web è definita attraverso HTML. Uno script JS si occupa del funzionamento della pagina, gestendo la richiesta dei dati al Backend e la visualizzazione degli stessi, attraverso i grafici, una volta ottenuta la risposta.

4.3.1 Dati visualizzati

La pagina permette innanzitutto di visualizzare la temperatura attuale presente all'interno dell'alveare, corrispondente all'ultimo valore inserito nel Database.

È poi possibile selezionare una data ed un intervallo orario per generare i due grafici relativi all'andamento della temperatura nel tempo e alle frequenze sonore medie nel periodo selezionato. Ricordiamo che le frequenze audio presentano variazioni già a partire da più di un mese prima della sciamatura, per cui notare particolari differenze tra due giornate può essere un campanello d'allarme.

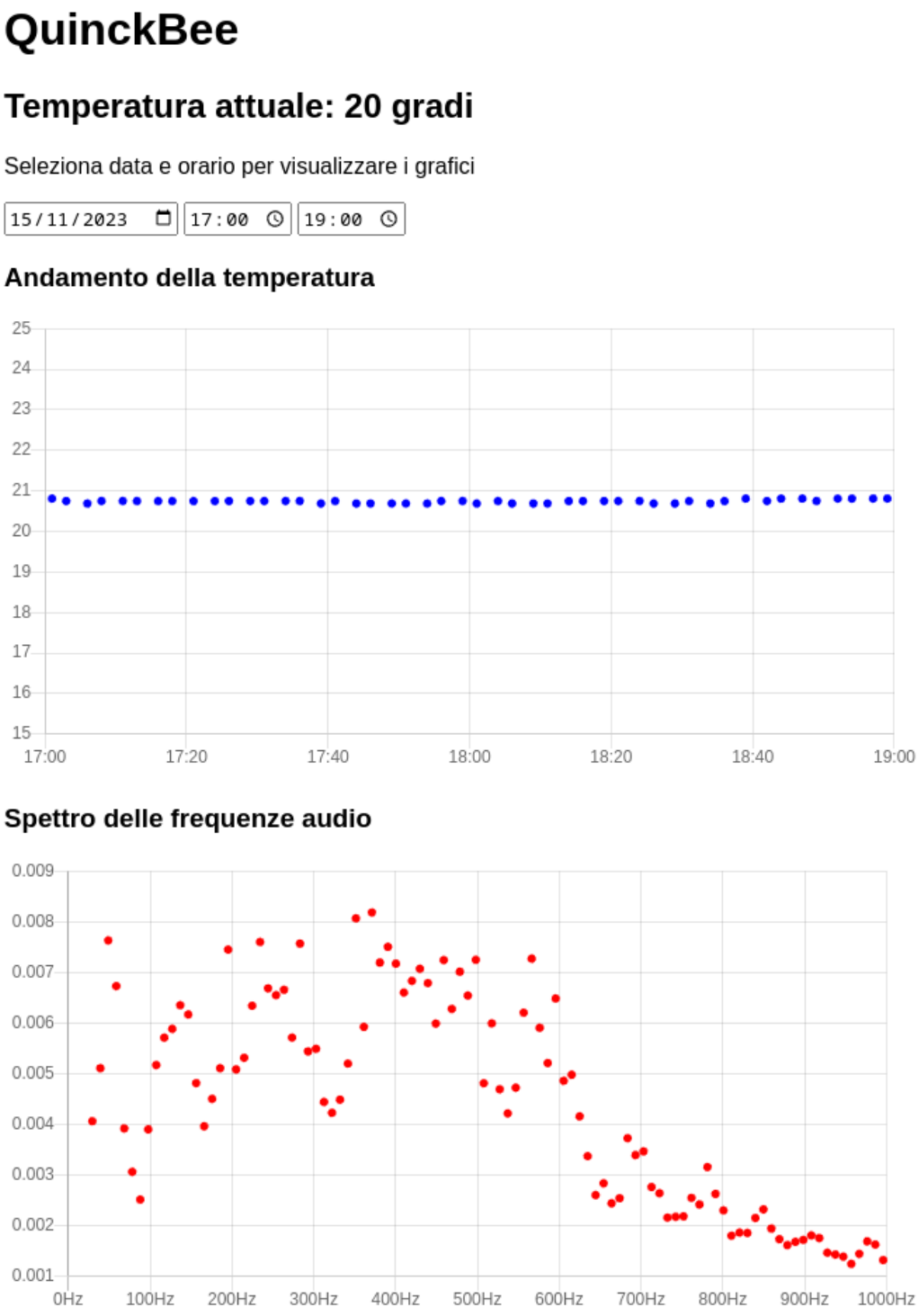

Figura 4.4: Pagina web del Frontend per la visualizzazione dei dati

4.3.2 Codice del Frontend

Struttura della pagina HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>QuinckBee</title>
    <script src="https://unpkg.com/axios/dist/axios.min.js"
   ></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart
   .js/2.9.4/Chart.js"></script>
    <script src="javascript.js"></script>
  </head>
  <body>
    <header>
      <h1>QuinckBee</h1>
    </header>
    <main>
      <section>
        <h2 id="current_temperature"></h2>
        <p>Seleziona data e orario per visualizzare i grafici
    </p>
        <input id="date_input" type="date">
        <input id="start_time" type="time" value="00:00">
        <input id="end_time" type="time" value="23:59">
        <h3>Andamento della temperatura</h3>
        <canvas id="temperature_chart"></canvas>
        <h3>Spettro delle frequenze audio</h3>
        <canvas id="frequency_chart"></canvas>
      </section>
    </main>
  </body>
</html>
```

Gli elementi `canvas` vengono utilizzati per disegnare i grafici per la visualizzazione dei dati.

Invio della richiesta HTTP al Backend

```
function getFrequencies(dateInput, startTimeInput,
    endTimeInput, frequencyChart) {
  let date = new Date(dateInput.value)
  let start_time = startTimeInput.value
  let end_time = endTimeInput.value
  axios({
    method: 'get',
    url: httpServer + '/frequency_values',
    withCredentials: false,
    params: {
      year: date.getFullYear(),
      month: date.getMonth(),
      day: date.getDate(),
      start_hour: start_time.substring(0, 2),
      end_hour: end_time.substring(0, 2),
    }
  }).then(function(response) {
    let data = response.data
    let values = data.map(element => {
      return {
        "x": element.frequency,
        "y": element.value
      }
    })
    ...
  })
}
```

Codice relativo alla richiesta HTTP effettuata dal Backend. I dati recuperati, racchiusi in un array, vengono mappati nella variabile `values` per poi essere mostrati nel grafico.

4.3.3 Librerie

- **Axios**[9]
  Questa libreria permette di gestire facilmente l'invio asincrono di richieste HTTP da ogni browser; è stata utilizzata in questo progetto per effettuare tutte le richieste dei dati al data loader.

- **Chart.js**[10]
  Libreria semplice ma molto versatile che permette di creare vari tipi di grafici, utilizzando l'elemento canvas di HTML e uno o più dataset.

[9] `https://github.com/axios/axios`
[10] `https://github.com/chartjs/Chart.js`

4.3.4 Deployment

Per fare in modo che la pagina web sia disponibile ovunque si è nuovamente fatto uso di Docker e della macchina virtuale EC2. Per realizzare il **web server** è stato utilizzato un container Docker creato con l'immagine **nginx**: si tratta di un software open-source che fornisce servizi di web serving, reverse proxying, caching, load balancing e non solo. In questo caso, la sua funzione è quella di fornire la pagina web ad un indirizzo IP statico, rendendola così visualizzabile attraverso qualunque web browser.

4.4 Implementazione dell'HiveSubsystem all'interno dell'alveare

L'ultimo passo della fase di implementazione prevede la trasformazione del circuito dell'HiveSubsystem, in modo tale da renderlo resistente alle condizioni ambientali e pronto al posizionamento all'interno dell'alveare. Innanzitutto, verrà rimossa la breadboard e saranno sostituiti i connettori con dei cavi impermeabili, saldati nei punti di collegamento. È stato anche deciso di sostituire il power-bank da 10.000mAh con uno nuovo da 30.000mAh: oltre a fornire una capacità triplicata, esso è provvisto di pannelli per la ricarica solare ed è waterproof.

Sulla base delle misure del microcontrollore è stata infine progettata una scatola che servirà a contenerlo e proteggerlo dalla pioggia.

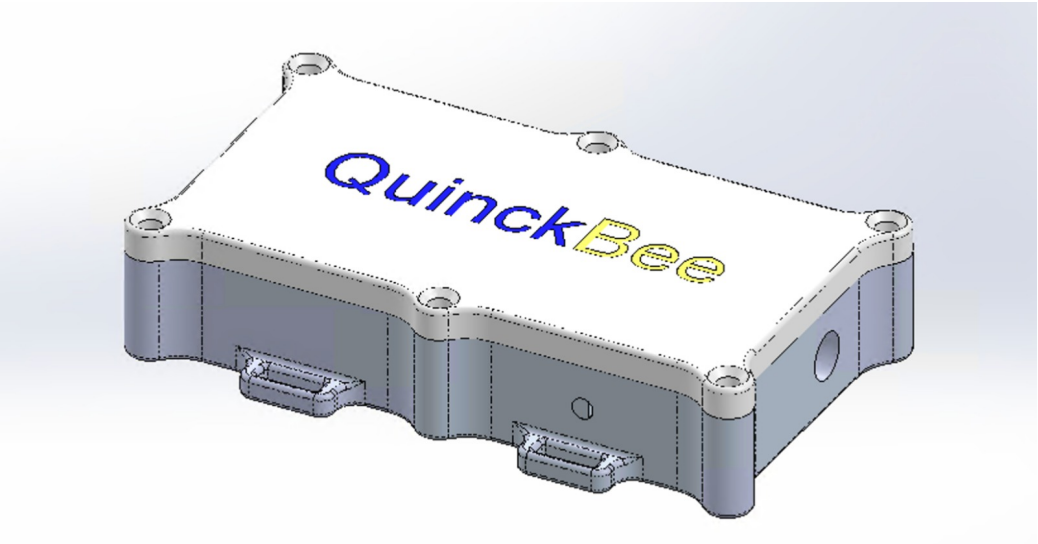

Figura 4.5: Scatola di contenimento del microcontrollore

La scatola è stata progettata per poter essere sigillata, tramite l'utilizzo di 6 viti, ed impedire l'infiltrazione della pioggia. Sono presenti due fori, per permettere il collegamento con la batteria esterna e con i sensori, ed è possibile fissare la scatola ad un piano o una parete, tramite l'uso di altre viti.

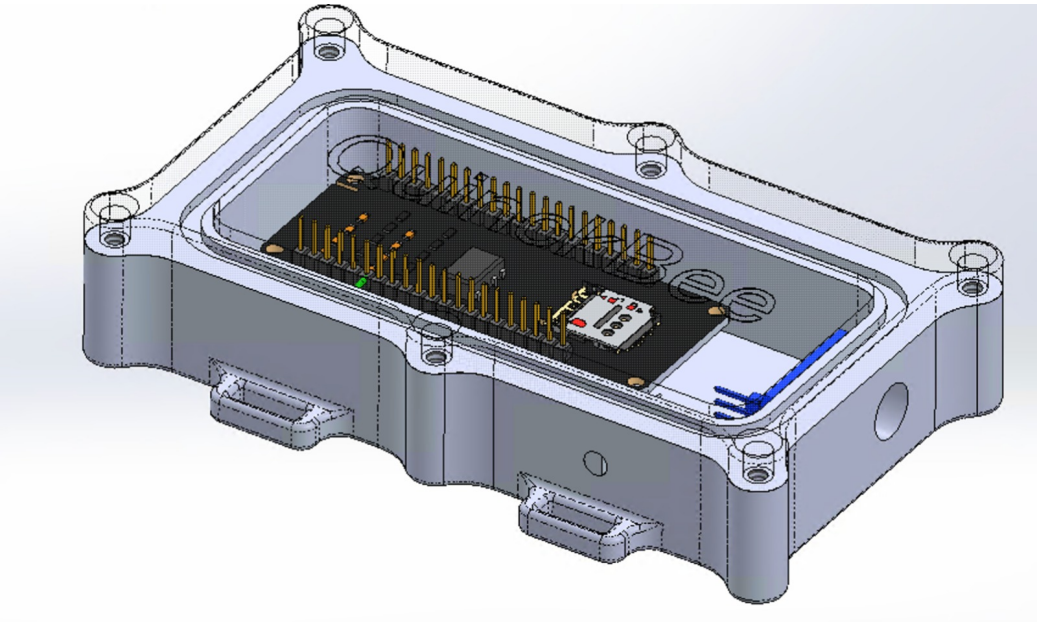

Figura 4.6: Posizionamento del microcontrollore all'interno della scatola

Il sistema è quindi pronto per essere posizionato nell'alveare e iniziare la sua fase di esercizio.

Capitolo 5

Discussione dei risultati ottenuti

5.1 Raggiungimento dei requisiti

Sono sicuramente stati raggiunti gli obiettivi di memorizzazione e visualizzazione dei dati definiti in fase di analisi. È stata verificata la veridicità delle frequenze sonore visualizzate facendo ascoltare al microfono il suono prodotto da una frequenza fissa di 440Hz.

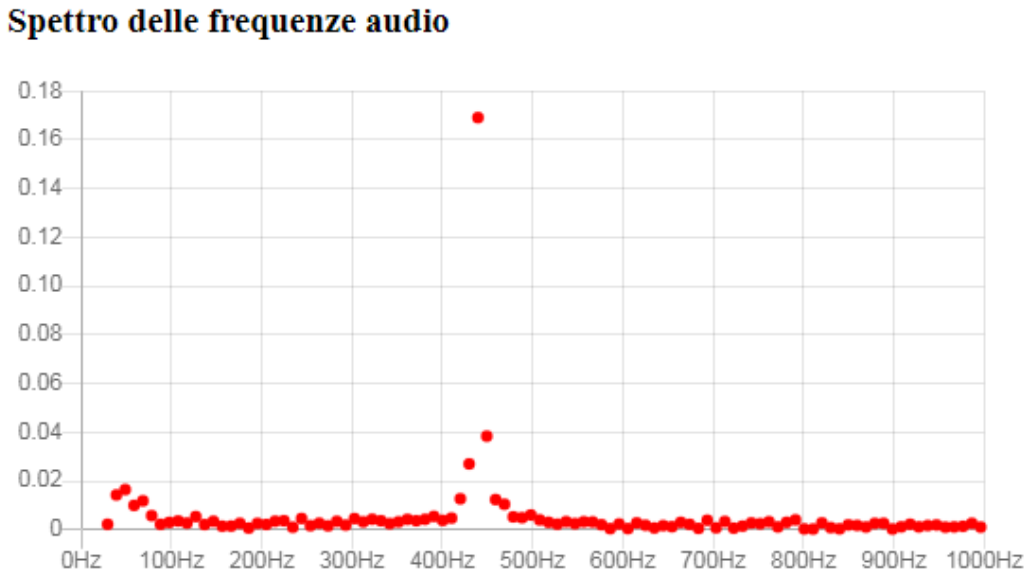

Figura 5.1: Spettro dell'audio della frequenza di 440Hz

Si sono trovate soluzioni alla mancanza di connessione wi-fi e di alimentazione elettrica, ed è stato fatto il possibile per proteggere il circuito dalla pioggia e dalle altre condizioni atmosferiche. Con il tempo si potrà monitorare l'operatività del sistema, controllando la raccolta dei dati, ed intervenire in caso si riscontrino dei problemi.

Per quanto riguarda il consumo energetico, considerando un consumo di 50mA in fase di esecuzione (secondo i dati relativi alla modalità modem-sleep di esp32) e la capacità del power-bank utilizzato di 30.000mAh, possiamo calcolare una durata della batteria teorica:

$$30.000mAh/50mA = 600h$$

Ovvero 25 giorni. In realtà, bisogna considerare che:

- il circuito si troverà per la maggior parte del tempo in modalità di risparmio energetico, con un consumo molto minore;
- il consumo di 50mA usato nel calcolo è comunque pessimistico;
- il power-bank si ricaricherà (se non completamente, almeno in parte) grazie ai pannelli solari a disposizione.

È chiaro, quindi, che la durata effettiva della batteria sarà di gran lunga maggiore, e potrà comunque essere monitorata periodicamente.

5.2 Utilità del sistema

Il sistema realizzato permette di monitorare da remoto un alveare attraverso due importanti misurazioni che, se interpretate correttamente, possono consentire di identificare i fenomeni di sciamatura. L'importanza di ciò risiede nel fatto che in questo modo è possibile controllare lo stato dell'alveare senza il bisogno di ispezionarlo frequentemente di persona.

Innanzitutto, ciò equivale a semplificare, per quanto possibile, il lavoro dell'apicoltore: capita spesso, infatti, che gli alveari siano collocati in luoghi sparsi e lontani tra loro, e poter ridurre il più possibile il numero di visite necessarie per controllare l'alveare vuol dire ridurre il tempo perso per raggiungerli.

Il secondo enorme vantaggio, invece, è che in questo modo le api e il loro ambiente vengono disturbati molto meno frequentemente: abbiamo già visto come questo abbia un grande impatto sul comportamento delle api e sullo stress da loro subito, per cui ciò non può che aiutare la sopravvivenza e la salvaguardia di questa specie.

5.3 Sviluppi futuri

Miglioramento dell'interfaccia grafica

In futuro, è possibile migliorare l'interfaccia grafica del Frontend, soprattutto per quanto riguarda i grafici. Sarebbe utile riuscire a fornire uno strumento di analisi migliore soprattutto per quanto riguarda le frequenze sonore.

Identificazione automatica della sciamatura

Sarabbe sicuramente interessante proseguire lo sviluppo del sistema implementando algoritmi per l'identificazione automatica degli eventi sciamatori.

Ciò potrebbe essere fatto sia analizzando la temperatura, con la possibilità quindi di informare l'apicoltore attraverso una notifica, che analizzando le frequenze sonore, con la possibilità in questo caso di avvisare in anticipo, attraverso la pagina web, di una possibile futura sciamatura.

In ogni caso, per sviluppare algoritmi di questo tipo occorre uno studio approfondito delle misurazioni raccolte durante la stagione primaverile, periodo in cui generalmente gli sciami di api tendono a presentare fenomeni di sciamatura. Il sistema verrà quindi mantenuto in funzione almeno fino alla fine dell'estate 2024, e sarà poi possibile procedere con l'analisi dei dati raccolti e quindi con lo sviluppo di algoritmi automatizzati.

Implementazione del sistema in più alveari

In futuro è possibile implementare questo sistema su altri alveari, raccogliendo dati relativi a varie famiglie di api.

In questo caso, sarebbe necessario associare a tutti i dati raccolti un identificatore dell'alveare di provenienza, e attraverso la pagina web si selezionerebbe l'alveare di cui si vogliono visualizzare i dati.

Maggiore sicurezza

Se si vuole una maggiore riservatezza dei dati, può essere implementato un sistema di login per verificare l'identità di chi cerca di visualizzarli.

Inoltre, potrebbe essere implementato un sistema di autenticazione sulle connessioni HTTP e MQTT per impedire a terzi, per esempio, di inserire dati non veritieri sul Database.

Capitolo 6

Conclusioni

In conclusione, devo dire di essere molto soddisfatto del lavoro svolto: sono stati soddisfatti i requisiti definiti in fase di analisi e il sistema, per quanto in uno stato ancora prototipale, sembra presentare basi solide per poter proseguire lo sviluppo in futuro.

Lavorare a questo progetto è stato per me molto interessante, perché mi ha permesso di ottenere una visione molto più completa riguardo a come funzionino i sistemi IoT di questo genere, sempre più diffusi oggi, e di approfondire argomenti come il campionamento dell'audio e le frequenze sonore o il deployment di applicazioni server.

Ci sono stati, durante lo sviluppo della tesi, anche momenti di difficoltà e vari imprevisti, soprattutto a causa delle numerose tecnologie utilizzate ed interfacciate tra loro. Devo ammettere di aver passato dei momenti di particolare stress, e mi sono sentito a volte un pò perso riguardo a come proseguire, ma alla fine di questo percorso sono molto contento e soddisfatto di aver trovato una soluzione ai problemi affrontati e di avere portato a termine questo progetto.

Bibliografia

[1] Redazione Ansa. "Giornata mondiale delle api, in Italia sono 100 miliardi". In: *Ansa.it* (2023). URL: https://www.ansa.it/canale_ambiente/notizie/acqua/2023/05/19/giornata-mondiale-delle-api-in-italia-sono-100-miliardi_2183deaf-11b4-44a6-8d4f-9625407a2c95.html#:~:text=L'ultimo%20censimento%202022%20indica,in%20500%20milioni%20di%20euro..

[2] Ferdinando Gagliotti. "Le nostre api sono in grave pericolo: ecco come possiamo aiutarle". In: *Il Mattino* (2023). URL: https://www.ilmattino.it/pelo_e_contropelo/le_nostre_api_grave_pericolo_possiamo_aiutarle-7338285.html?refresh_ce.

[3] Michael-Thomas Ramsey et al. "The prediction of swarming in honeybee colonies using vibrational spectra". In: *scientific reports* (2020). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7298004/#CR17.

[4] Aleksejs Zacepins et al. "Remote detection of the swarming of honey bee colonies by single-point temperature monitoring". In: *Biosystems engineering* (2016). URL: https://www.sciencedirect.com/science/article/abs/pii/S1537511016300964.

[5] Aleksejs Zacepins et al. "When It Pays to Catch a Swarm—Evaluation of the Economic Importance of Remote Honey Bee (Apis mellifera) Colony Swarming Detection". In: *Agriculture* (2021). URL: https://www.researchgate.net/publication/355066136_When_It_Pays_to_Catch_a_Swarm-Evaluation_of_the_Economic_Importance_of_Remote_Honey_Bee_Apis_mellifera_Colony_Swarming_Detection.